Operating System Project - Phase 2 (Process and Memory Management)

Prerequisite

Phase 2 is built on phase 1. For this phase, it is required that you complete the implementation of instruction set from phase 1.

What to do?

- Creating process control block
- Maintaining ready and running queues
- Memory allocation (segmentation with paging) & deallocation
- Loading, running and terminating process
- Error Handling
- CPU Scheduling (round-robin scheduling)

Process Control Block

Every process has a process control block (PCB), which has all the information required to run a process. PCB contains at least the following:

- Process ID
- Process priority
- Process size (code+data+segment)
- Process File name
- General Purpose Registers
- Special Purpose Registers
- Page Table

Program Loading

A process is read from a file. When a process is loaded

- It is parsed for valid priority, valid data and code sizes. No need to check instruction syntax at this stage.
- Priority ranges from 0-31. Any priority less than 0 or greater than or equal to 32 is invalid
- The program is loaded into memory.
- PCB is created and added to the ready queue.

Data loading

Data will be loaded in memory according to the size specified. For example if data size is defined as '**0004'**, it means that data size is 4 bytes. Only four bytes will be loaded in the data segment.

Code loading

Only that much code is read which is specified in code segment declaration. For example if code size is **'000e'** then code size is 14 bytes and only fourteen bytes will be read from the file and loaded in code segment.

You should be able to load more than one program in memory and maintain ready queues of the processes.

Program Execution

The first process is selected and removed from the ready queue and its state is restored (its registers from PCB are restored to the CPU registers). Instructions are fetched from memory, decoded, executed and if required, values written back. Syntax errors are checked at process execution stage. Any invalid instruction or error condition terminates the process. During execution, a process will move between running and ready queues.

Program termination

When the process finishes, either due to abnormal conditions or reaching the 'end' statement, its PCB is removed from the queue and its memory dump (data, stack and code sections) and PCB are printed.

Errors

An error terminates the process.

- Invalid opcode / register code
- Trying to access data outside allocated space
- Trying to jump, using call or branch statement, to area outside the allocated space
- Invalid priority (<0 or >31)
- Stack overflow/underflow
- Divide-by-zero
- Invalid code size (code size specified is less than or greater than the actual code)

1. Multiple Process Management

The salient features of Multiple Process Management are:

- Maintaining ready & running queues
- CPU Scheduling (round robin scheduling algorithm)

Ready & Running Queues

Ready and running queues are implemented as priority queues of PCBs. There will be only one process in running queue whereas ready queue may have more than one.

CPU Scheduling

Implement Round-robin scheduling algorithm can be used with 4 cycle slice [Quantum=4, it should be configurable] (2 instructions are executed in each time slice).

Assumption: Each instruction takes 2 clock cycles.

2. Memory management

What to do?

- Memory Allocation & De-allocation
- Trapping access violations

Segmentation with paging has to be used to address the memory. The Memory has to be divided into pages of size 1K Bytes. You would need the following data structures:

- *Segment tables – Segment tables for each segment of a process(We're assuming there are only 3 segments : code, data and stack)*
- *Page tables* – Page tables for all process segments in the memory
- *Page frames* – Divide user memory in frames
- *Free page frame list* – List of all page frames not currently in use by any process